A Hybrid Approach to Reverse Engineering on Combinational Circuits

Wuqian Tang [1], Yi-Ting Li[1], Kai-Po Hsu[1], Kuan-Ling Chou[1], You-Cheng Lin[1], Chia-Feng Chien[1], Tzu-Li Hsu[1], Yung-Chih Chen[2], Ting-Chi Wang[1], Shih-Chieh Chang[1], TingTing Hwang[1], and Chun-Yao Wang[1]

[1]National Tsing Hua University, Taiwan, ROC
[2]National Taiwan University of Science and Technology, Taiwan, R.O.C.

***Abstract*—Reverse engineering is a process that converts low-level description to high-level one. In this paper, we propose a hybrid approach consisting of structural analysis and black-box testing to reverse engineering on combinational circuits. Our approach is able to convert combinational circuits from gate-level netlist to Register-Transfer Level (RT-level) design accurately and efficiently. We developed our approach and participated in Problem A of the 2022 CAD Contest @ ICCAD. The revised version of our program successfully converted most cases and achieved higher scores than the 1^{st} place team in the contest.**

I. Introduction

With the advances of process technology, modern designs can incorporate billions of logic gates. However, as the design complexity grows, detecting hidden trojans [11] or backdoors [12] becomes increasingly challenging. Given highly complex gate-level netlists, many methods have been proposed to address the issues of detecting trojans and backdoors [5], [6], [13]. This area of applications can be based on reverse engineering [5]. If combinational circuits can be converted from gate-level to RT-level, designers would easily understand the content of circuits. Thus, there is a pressing need to extract higher level behavior of a circuit from the gate-level. To address the problem of reverse engineering on combinational circuits, many methods have been proposed [7]–[10].

However, there is a noticeable shortcoming in the previous research on reverse engineering of combinational circuits. Traditional methods often rely on structural analysis, such as XOR-Tree [1]. While these methods can effectively identify basic arithmetic operators, they fail when dealing with complex arithmetic operations, or those involving bit shifting or exponent. In this paper, we propose a hybrid approach that combines structural analysis and black-box testing for efficient combinational circuit reverse engineering. It aims to convert a combinational circuit from gate-level netlist to RT-level description in Verilog format [4]. As compared to the performances of the 1^{st} place team in Problem A of the 2022 CAD Contest @ ICCAD [2], our approach achieves higher scores than their result in the contest.

This work is supported in part by the National Science and Technology Council (Taiwan) under MOST 109-2221-E-007-082-MY2, MOST 111-2221-E-007-121, MOST 111-2221-E-011-137-MY3, NSTC 112-2218-E-007-014, NSTC 112-2221-E-007-106-MY2, NSTC 112-2221-E-007-108, and NSTC 112-2425-H-007-002.

II. The Proposed Approach

Our approach begins by extracting Coefficients-Undetermined Conditional Operatoins (CUCOs) and Coefficients-Undetermined Arithmetic Operations (CUAOs) from the gate-level netlist using structural analysis. Following this, we employ black-box testing to conduct random simulation on the circuit. Lastly, we utilize the Gaussian elimination technique to resolve undetermined coefficients in the CUCOs and CUAOs.

A. Structural Analysis

1) Extracting CUAOs: For each output vector, we extract a corresponding CUAO. For example, assume that we have an output vector of length 4, denoted as out[3:0]. We may determine a set of input vectors, in_1, in_2, in_3, in_4, etc, that constitute the functional support variables for each wire in out. The example can be illustrated as:

$$out = f(in_1, in_2, in_3, in_4, ...) \tag{1}$$

where the input vectors determine the value of out, and f denotes an arithmetic operation to be extracted.

To extract the CUAO, we reorganize each wire in the out into the Exclusive-Sum Of Products (XSOP) form. For instance, suppose the XSOP form of $out[0]$ is:

$$out[0] = (in_1[0] \wedge in_2[0]) \ \oplus \ (in_3[0] \wedge in_4[0]) \tag{2}$$

where $\wedge$ is the bit-and operator and $\oplus$ is the bit-xor operator. We can deduce the multiplication terms by collecting the product terms in EQ (2). The corresponding CUAO is:

$$f = Coe_1 \times in_1 \times in_2 + Coe_2 \times in_3 \times in_4 + Const \tag{3}$$

By performing the XSOP extraction operation from Least Significant Bit (LSB) to Most Significant Bit (MSB) of out, more terms can be added into f.

2) Extracting CUCOs: For arithmetic operations constituted of addition, subtraction, and multiplication, the lower bit can influence the higher bit, but the converse is not held. For instance, consider a 4-bit adder defined as $sum[3:0] = a[3:0] + b[3:0]$, $sum[0]$ is determined by $a[0]$ and $b[0]$. Due to carry propagation, $sum[1]$ is determined by $a[0]$, $b[0]$, $a[1]$, and $b[1]$. The subsequent bits follow the same rule. Assume that we possess a 4-bit input vector c, and the arithmetic operations of this sum are encompassed in the condition $if(c > 0)$, then every bit of sum's functional support variables will include $c[0]$ to $c[3]$.

979-8-3503-4859-0/DATE24/© 2024 EDAA

By employing this principle, we can identify input vectors that may appear in a conditional operation. Here, we explore combinations of input vectors in the conditional operation. For instance, if we know that both $c1$ and $c2$ could be present within a conditional operation, an initial attempt would be: $if\ (Coe_1 \times c_1 + Coe_2 \times c_2 \circledcirc Const)$, where $\circledcirc$ denotes any comparison operator. When the subsequent steps do not produce a valid RT-level design, our method will be adapted to explore more intricate combinations such as $if\ (Coe_1 \times c_1 \times c_1 + Coe_2 \times c_1 \times c_2 + Coe_3 \times c_2 \times c_2 \circledcirc Const)$, and so on. This iterative procedure continues until the accurate RT-level design is discovered.

B. Black-box Testing

For any CUAO and CUCO, it can be represented as:

$$out = Const + Coe_1 \times P_1 + Coe_2 \times P_2 + Coe_3 \times P_3 + \dots \quad (4)$$

Each P_i denotes a term constructed from input vectors and non-additive arithmetic operations. By assigning a variety of random patterns to the circuit, we have a set of output vectors with corresponding values. This process results in a system of linear equations. Subsequently, we can employ Gaussian elimination to determine all values of the undetermined coefficients.

III. Experimental Results

TABLE I: The Results of Our Approach.

Cases	$Cost_{ori}$	$Cost_{RTL}$		Reduction (%)	CPU (s)		
		Ours	1^{st}		Ours	1^{st}	$\downarrow$ (%)
Test01	65	2	2	96.9	0.012	0.137	91.2
Test02	81	2	2	97.5	0.009	0.126	92.9
Test03	453	3	3	99.3	0.015	0.141	89.4
Test04	8196	6	6	99.9	0.372	0.473	21.4
Test05	1724	6	6	99.7	0.056	0.212	73.6
Test06	5310	3	3	99.9	0.151	0.431	65.0
Test07	2961	9	9	99.7	0.168	0.389	56.8
Test08	3302	4	4	99.9	0.112	0.227	50.7
Test09	7988	3	3	100.0	0.208	0.537	61.3
Test10	3029	4	4	99.8	0.119	0.249	52.2
Test11	2003	2	2	99.9	0.026	0.238	89.1
Test12	13290	23	23	99.8	0.920	1.047	12.1
Test13	1067	18	18	98.3	0.244	0.728	66.5
Test14	107	2	2	98.1	0.051	0.122	58.2
Test15	4231	12	12	99.7	0.692	0.782	11.5
Test16	**999**	**5**	**6**	**99.4**	**0.227**	**0.232**	**2.2**
Test17	1365	7	7	99.5	0.243	1.479	83.6
Test18	174	12	12	93.1	0.115	0.138	16.7
Test19	2379	91	91	96.2	0.228	0.234	2.6
Test20	5104	18	18	99.6	0.594	1.760	66.3
Test21	**2074**	**59**	**93**	**97.2**	**0.732**	**0.890**	**17.8**
Test29	**842**	**53**	**-**	**93.7**	**5.973**	**-**	**-**
Test30	16525	24	24	99.9	0.915	1.600	42.8
Avg.	-	-	-	-	-	-	51.1

We implemented the proposed approach in C++ and conducted experiments on a Linux platform (Ubuntu 20.04.03 LTS) with an AMD EPYC 7282 CPU (2.8 GHz) and 256 GB of RAM. The benchmarks for the experiments were sourced from Problem A of the 2022 CAD Contest @ ICCAD [3]. Each benchmark is a gate-level netlist synthesized from RT-level design composed of operators like addition, subtraction, multiplication, conditioning, and/or operators, bit selection, and concatenation. The experimental results are summarized in Table I, where we also compare our result with the one of 1^{st} place team of the contest. The columns $Cost_{ori}$ and $Cost_{RTL}$ represent the $Cost$ [3] of the original gate-level netlist and that of the RT-level design converted using our approach, respectively. In general, a smaller $Cost$ indicates a higher level of description of the derived RT-level design, and thus higher readability. The column $Reduction(\%)$ represents the reduction ratio of $Cost_{RTL}$ relative to $Cost_{ori}$. The $CPU(s)$ column measured the runtime of the program execution. We have listed all the benchmarks that we successfully converted. The cases not successfully converted to RT-level design are denoted by "$-$". For benchmarks Test22 to Test28, both our approach and the 1^{st} place team failed to convert to RT-level design. As a result, these benchmarks are omitted from the table. As compared with the 1^{st} place team, the reduction ratio of our method is either superior to or equivalent to that of the 1^{st} place team across all benchmarks. Moreover, our approach, on average, required only 49% of the CPU time compared to the 1^{st} place team.

References

[1] M. Barbareschi, S. Barone, N. Mazzocca, and A. Moriconi, "A catalog-based aig-rewriting approach to the design of approximate components," *IEEE Transactions on Emerging Topics in Computing*, 2022.

[2] Y.-G. Chen, C.-Y. Wang, T.-W. Huang, and T. Sato, "Overview of 2022 cad contest at iccad," in *Proc. of the IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–3.

[3] C.-H. Chou, C.-J. Hsu, C.-A. Wu, and K.-H. Tu, "2022 cad contest problem a: Learning arithmetic operations from gate-level circuit," in *Proc. of the IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–4.

[4] P. M. Donald Thomas, *The Verilog® Hardware Description Language*. Springer New York, NY, 2002.

[5] M. Fyrbiak, S. Wallat, P. Swierczynski, M. Hoffmann, S. Hoppach, M. Wilhelm, T. Weidlich, R. Tessier, and C. Paar, "Hal—the missing piece of the puzzle for hardware reverse engineering, trojan detection and insertion," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 3, pp. 498–510, 2018.

[6] Z. Huang, Q. Wang, Y. Chen, and X. Jiang, "A survey on machine learning against hardware trojan attacks: Recent advances and challenges," *IEEE Access*, vol. 8, pp. 10 796–10 826, 2020.

[7] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik, N. Shankar, and S. A. Seshia, "Wordrev: Finding word-level structures in a sea of bit-level gates," in *Proc. of the IEEE International Symposium on Hardware-Oriented Security and Trust*, 2013, pp. 67–74.

[8] W. Li, Z. Wasson, and S. A. Seshia, "Reverse engineering circuits using behavioral pattern mining," in *Proc. of the IEEE International Symposium on Hardware-Oriented Security and Trust*, 2012, pp. 83–88.

[9] T. Meade, S. Zhang, and Y. Jin, "Netlist reverse engineering for high-level functionality reconstruction," in *Proc. of the IEEE Asia and South Pacific Design Automation Conference*, 2016, pp. 655–660.

[10] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. Y. Tan, A. Tiwari, N. Shankar, S. A. Seshia, and S. Malik, "Reverse engineering digital circuits using structural and functional analyses," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 63–80, 2013.

[11] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[12] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *Proc. of the IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 15–19.

[13] Y. Yang, J. Ye, Y. Cao, J. Zhang, X. Li, H. Li, and Y. Hu, "Survey: Hardware trojan detection for netlist," in *Proc. of the IEEE Asian Test Symposium*, 2020, pp. 1–6.